A GENERIC PROGRAMMABLE ARBITER WITH DEFAULT MASTER GRANT

Frédéric Pétrot and Denis Hommais

Département ASIM du LIP6
Université Pierre et Marie Curie
Paris, France

ABSTRACT

This paper details the design and implementation of a centralized bus arbiter implementing programmable fixed priorities arbitration. The arbiter also handles default master grant to the master with highest priority. The arbitration algorithm is computed using a tree of specialized comparators to fully exploit hardware parallelism. The design is implemented as a generic VHDL model whose parameter is the number of masters. After synthesis and place & route, a 16 masters arbiter has a critical path delay of 7.5 ns in 0.5 μm technology.

1. INTRODUCTION

On chip buses have received much attention from the industrial world in the past few years, and several initiatives have proposed bus protocol standards: the PI-Bus [1] of OMI, the Amba bus [2] of ARM, the FISPbus [3] of Mentor Graphics, and others. More recently, a abstract protocol, called VCI[4], has been devised by the OCB group of the VSI Alliance.

On a bus, the implementation of the protocol lies from one side in the master –the initiator of a transaction– and slave –the target of a transaction– component plugged on the bus and from an other side in the Bus Control Unit.

The Bus Control Unit is responsible of slave selection, timeout handling, protocol consistency verification and master arbitration. Arbitration means choosing a unique master among several masters requesting the bus.

The choice of an arbitration scheme is usually left to the implementation, as it depends much of the application. Most standards do not give any specifications for arbitration, the PCI [5] being a notable exception in asking for a « fair » algorithm.

Our goal in this paper is not to devise a new bus allocation scheme, but to cover the implementation of programmable fixed priorities arbitration schemes. Theses schemes are of practical importance in many applications. The most important ones concerning real-time constrained systems. Programmable fixed priorities are needed to implement deadline first scheduling for a given set of tasks[6] to maximize bus utilization. They can be extended quite easily to the handling of mixed random/flow data exchanges that has been devised in [7] to minimize bus access latencies. They are also relevant to reconfigurable systems in the which the set of tasks to execute changes over time.

The other important point of this study is the support of the default master grant that exists in many standards[1, 8, 2]. The default grant mechanism allows the default master to get the bus before requesting it, gaining a clock tick. The default master should be the most constrained master –for example a processor instruction cache–. This is very simple to implement for the master: it simply checks that it is granted the bus prior to request it, and if so bypasses a request state. Unfortunately, for the Bus Control Unit, this means being able to know if the bus requester is the default master, and this requires a complex computation.

These points are trivial if the priorities are hardwired, but gain an order of magnitude in complexity when they are programmable.

Section 2 formulate the problem both in a natural and more theoretical way. The details of the design and implementation are presented in Section 3, and the results in Section 4. We conclude Section 5.

2. PROBLEM STATEMENT

Although the problem is of practical importance since a long time, very few publications have dealt with the design of programmable fixed priority arbiters. Most commercially available bus arbiters perform simple hardwired fixed priorities[9, 10] or round-robin arbitration[10, 8]. [11] details a complex round-robin arbitration mixed with priority levels to implement biased-fairness. It however does handle only two bits for priority values, and no details are given on the implementation.

Our goal is to devise a general architecture for programmable fixed priority arbiters with a variable number of masters. To be consistent with the fixed priority scheme, the default master must be the master currently with the highest priority. Technological constraints impose an upper bound on this number n, but there is no theoretical reason for this limit.

The arbiter has n request lines Req_i, n grant lines Gnt_i,

This work has been funded in part by the COSY 25443 Esprit Project.

and n priority lines $Prio_i$ encoded on $\lceil \log_2(n) \rceil$ bits, $0 < i < n$. The $Prio_i$ lines can take any value in $\{0, \ldots, 2^{\lceil \log_2(n) \rceil} - 1\}$, and are supposed to be stable during the whole clock period. There are indeed output of registers whose values are set by the software controlling the arbiter. Therefore, there can be no warranty that all values are distinct, and we must handle priority ties. The algorithm of the arbiter with default master grant is as follow.

1. The master with highest priority among the requesting masters is granted the bus. If more than one master are requesting and have the highest priority, then there is a tie. We break it arbitrarily in granting the bus to the master with the smallest request line number,
2. The highest priority masters is granted the bus if no requests are pending. Here also, several masters may have the highest priority. In such cases, we break the tie as above.

We have the further constraint that the arbitration must take place in a single bus cycle.

To arbitrate among the requesting masters necessitate to have the non-requesting master priorities set to the lowest possible value 0. Since the priorities are set by software, we must ensure that a non requesting master will never have a higher priority than a requesting master with the lowest priority. Since the software can write values between 0 and $2^{\lceil \log_2(n) \rceil} - 1$ inclusive in the $Prio_i$ registers, any requesting master must have an internal priority as seen by the arbiter of at least $2^{\lceil \log_2(n) \rceil}$. This is easily achieved by setting the internal priority as $Req_i \| Prio_i$, where $\|$ is the concatenation operator. A requesting master of priority 0 will actually have a priority of $2^{\lceil \log_2(n) \rceil}$. Any non-requesting master will actually have a priority of at most $2^{\lceil \log_2(n) \rceil} - 1$.

To handle the default master case, we remark that the highest priority master grant line is set when *all* masters are requesting and this is exactly what we want when *no* masters are requesting. So we simply detect that no requests are pending, and set all requests as pending at the input of the arbiter. This translates to $1 \| Prio_i$ for the internal priorities.

Using the above remarks, the problem is specified more formally –see Algorithm 1– in order to be able to devise a hardware architecture.

3. DESIGN AND IMPLEMENTATION

We assume that the Req_i signals are available early on the bus. Most busses demand that these signal be stable within 20% of the bus clock period. The arbiter has tight timing constraints because it may well be on the critical path of the system, thus the hardware parallelism must be fully exploited.

We note the bit b of signal X at depth d in the tree as $X_{b,d}$.

Algorithm 1 Computation of the grant signals

if $\bigvee_{0<i<n} Req_i = 1$ **then** *At least one request is pending*
  $\mathcal{H}_r = (j | \forall_{0<k<n} Req_j \| Prio_j \geq Req_k \| Prio_k)$
    $\mathcal{H}_r$ is the ordered set of highest priority masters currently requesting
  $r = \min(\mathcal{H}_r)$
    r is the first element of $\mathcal{H}_r$, i.e. *the highest priority requester with the smallest request line index*
  $Gnt_r = 1$
  $Gnt_k = 0, \forall k | 0 < k < n, k \neq r$
else *No requests are pending*
  $\mathcal{H}_1 = (j | \forall_{0<k<n} 1 \| Prio_j \geq 1 \| Prio_k)$
    $\mathcal{H}_1$ is the ordered set of highest priority masters
  $r = \min(\mathcal{H}_1)$
    As above, r is the first element of $\mathcal{H}_1$, i.e. *the highest priority master with the smallest request line index*
  $Gnt_r = 1$
  $Gnt_k = 0, \forall k | 0 < k < n, k \neq r$
end if

The input stage of the arbiter, depicted in Figure 1(b), is necessary to handle the default master case. We detect if there are requests in computing $Req = \bigvee_{0 \leq i < n} Req_i$, and if no requests are pending, we force all $R_{i,0} = 1$.

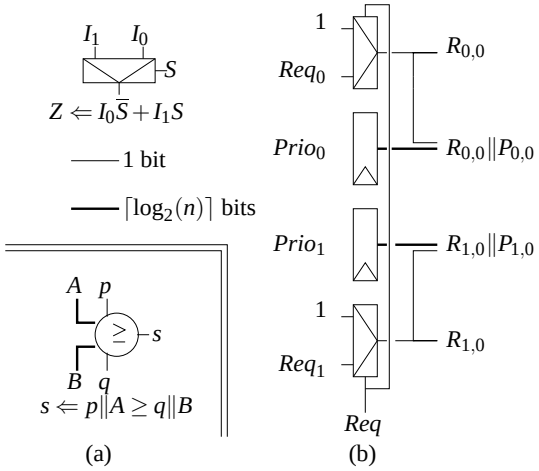

Figure 1: (a) Comparator (b) Arbitration tree input stage

Beyond this stage, we do not need to know if we are computing to grant the default master or any other master.

The arbitration algorithm is implemented as a binary tree, and has therefore a depth of $\lceil \log_2(n) \rceil$. The basic element of the tree is a compator of $\lceil \log_2(n) \rceil + 1$ bits. As shown Figure 2, this element is generic because its function depend on its depth in the tree.

At depth d, this comparator computes two values for depth $d+1$:

1. A boolean value for every $R_{i,d+1}$ signals,

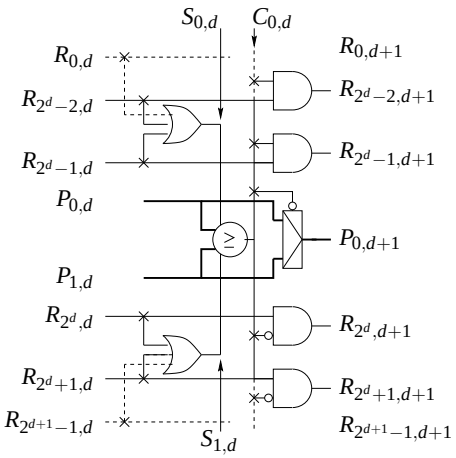

Figure 2: Basic generic element of the arbitration tree

2. A priority that is the maximum of the priorities of the preceding stage.

At depth d, there are four possible cases.

(1) $S_{0,d} = 0$ and $S_{1,d} = 0 \Rightarrow R_{i,d+1} = 0$ for $0 \leq i < 2^{d+1}$, and the value computed for $P_{0,d+1}$ is unused afterwards.
(2) $S_{0,d} = 1$ and $S_{1,d} = 0 \Rightarrow P_{0,d+1} = P_{0,d}$, $R_{i,d+1} = 0$, for $2^d \leq i < 2^{d+1}$, and $R_{i,d+1} = R_{i,d}$ for $0 \leq i < 2^d - 1$.
(3) $S_{0,d} = 0$ and $S_{1,d} = 1 \Rightarrow P_{0,d+1} = P_{1,d}$, $R_{i,d+1} = 0$, for $0 \leq i < 2^d - 1$, and $R_{i,d+1} = R_{i,d}$ for $2^d \leq i < 2^{d+1}$.
(4) $S_{0,d} = 1$ and $S_{1,d} = 1 \Rightarrow P_{0,d+1} = \max(P_{0,d}, P_{1,d})$, and if $C_{0,d} = 1$, case (2) applies, otherwise case (3) applies.

From these, the properties that ensure the proper logical implementation of Algorithm 1 can be shown by induction.

(a) If there is at least one $R_{i,d} = 1$, there is one and only one $R_{i,d+1} = 1$. This proves that at the output, because we always have at least one *Req* line activated, one and only one *Gnt* will be activated,
(b) $P_{i/2,d+1}$, when used by a later stage, is $\max(P_{i,d}, P_{i+1,d})$. This in fact proves that at depth $d+1$, $P_{i/2^d,d+1} = \max(Prio_0, \ldots, Prio_{2^d})$.

Figure 3 illustrates the architecture for an 8 master arbiter.

Multi-media arbitration[7] can be build using two such arbiters. The primary idea behind this approach is to chose to serve either random requests, such as the one generated by cache misses, or continuous ones, such as the one issued by digital signal processing engines. The choice to serve either a random or a continuous request doesn't influence the design of the arbiter, and we direct the interested reader to [7] for details on this choice.

We have two independent sets of requests, random, $\mathcal{R}$, and continuous, $\mathcal{C}$. We perform arbitration within both sets to elect the granted master, that we call $Gnt_{\mathcal{R}}$ and $Gnt_{\mathcal{C}}$. Then, eight cases are possible to obtained the bus master. It depends on the current choice between the random, $r/\overline{c} = 1$, or continuous, $r/\overline{c} = 0$, masters and the fact that a granted master is granted by default or not.

$Gnt_{\mathcal{R}}$	$Gnt_{\mathcal{C}}$	Choice	Decision
Default	Default	Random	$Gnt_{\mathcal{R}}$
Default	Request	Random	$Gnt_{\mathcal{C}}$
Request	Default	Random	$Gnt_{\mathcal{R}}$
Request	Request	Random	$Gnt_{\mathcal{R}}$
Default	Default	Continuous	$Gnt_{\mathcal{C}}$
Default	Request	Continuous	$Gnt_{\mathcal{C}}$
Request	Default	Continuous	$Gnt_{\mathcal{R}}$
Request	Request	Continuous	$Gnt_{\mathcal{C}}$

This algorithm is quite simple, but needs to know if a granted master has been granted by default. This information is also required by the Bus Control Unit controller, so we shall now explain how to obtain it.

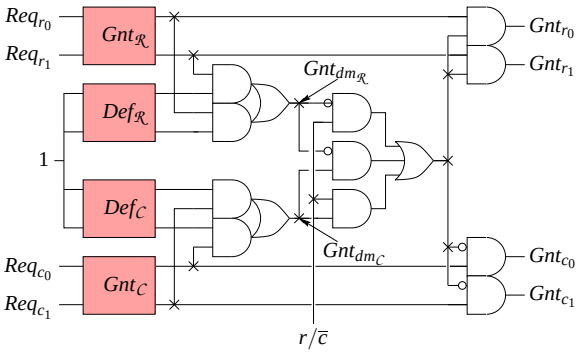

Figure 4: Random/continuous arbitration example

We need to compute through combinational logic the default master random master, $dm_{\mathcal{R}}$, and the default continuous master, $dm_{\mathcal{C}}$. They are computed using Algorithm 1 when no requests are pending because then we have $Gnt_{dm} = 1$. We cannot share the same hardware, but we can use the same module in the which we set all Req_i lines to '1'. The result are the vectors $Gdm_{\mathcal{R}}$ and $Gdm_{\mathcal{C}}$. We now check if the unique bit set in $Gnt_{\mathcal{R}}$ (in $Gnt_{\mathcal{C}}$) is equal to the unique bit set in $Gdm_{\mathcal{R}}$ (in $Gnt_{\mathcal{C}}$), by performing $\bigvee_{0<i<n} Gnt_{i_{\mathcal{R}}} \wedge Gdm_{i_{\mathcal{R}}}$ (and identically for continuous). Such an arbiter with 2 random and 2 continuous masters is presented Figure 4.

4. RESULTS

The programmable fixed priority arbiter has been implemented in generic VHDL. It is under the form of a soft IP block whose parameter is the number of masters.

We have synthesized the arbiter using Synopsys for 2, 4, 8 and 16 masters. We have extracted the critical path

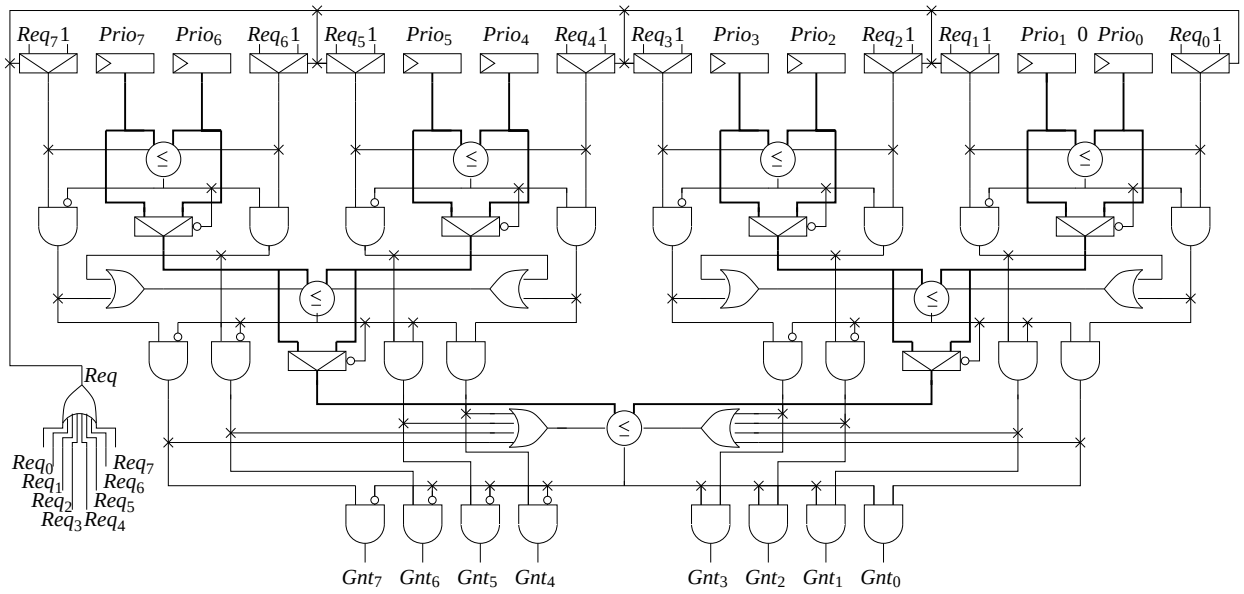

Figure 3: Example arbiter tree: 8 request lines

and area of the placed and routed circuits for a 0.5 μm technology using 3 levels of metal, 75°C, worst case transistor parameters. The following table details the delays and areas as a function of the number of masters.

Nb of masters	2	4	8	16
Delay (ns)	0.56	2.17	5.87	7.54
Area (μm^2)	63	7308	15120	118944
Gates	7	58	120	569

With this implementation, and assuming the Req_i signals are available within 20% and the Gnt_i signals must be available within 20% of the bus period, the arbiter runs at $\frac{10^9}{7.54+0.4\times7.54} \approx 95$ Mhz for 16 masters.

This still leaves room for driving long grant line to achieve around 80 MHz, which is the target frequency of industrial systems based on the PI-Bus, for example.

Busses running at 133 MHz, as required for example in the new PCI specification, are within the reach of this approach. If only 8 masters are necessary, the running frequency, computed with a 40% guard as above, is ≈ 120 Mhz. Scaling down the technology to 0.35 μm makes it possible to increase this number significantly.

5. CONCLUSION

In this paper, we have detailed the design of an arbiter implementing a programmable fixed priority scheme, also able to handle a default master grant. This study is of practical interest, as shown firstly by the requirements of real-time application and secondly by the current bus standardization initiative: arbitration is centralized and depends only on well agreed upon signals.

The architecture is generic, and allows to achieve around 110 MHz for 16 masters, a large upper bound on the acceptable number of masters on a single bus. We have also shown that handling a default master is costly in area for the Bus Control Unit, because an identical comparator tree has to be used to generate the signal that indicates that the current requesting master is the default master. This however doesn't incur a delay penalty, making this solution suitable for high performance systems.

6. REFERENCES

[1] Open Microprocessor systems Initiative, *DRAFT STANDARD OMI 324: PI-Bus Rev. 0.3d*, 1994.

[2] S. B. Furber, *ARM System Architecture*, ch. 8, pp. 234–238. Addison Wesley Longman, 1996.

[3] N. Thorne, "On-chip buses enable block based asic / fpga designs," in *Proceeding of IP'97 Europe*, (Bracknell, UK), Oct. 1997.

[4] VSI Alliance, *On Chip Bus Development*, June 1995. `http://www.vsi.org/library/specs/summary.htm#ocb`.

[5] PCI Special Interest Group, *PCI Local Bus Specification, Rev. 2.1*, June 1995.

[6] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, pp. 46–61, Jan. 1973.

[7] S. Hosseini-Khayat and A. D. Bovopoulos, "A simple and efficient bus management scheme that supports continuous streams," *ACM Transactions on Computer Systems*, vol. 13, pp. 122–140, May 1995.

[8] T. Shanley and D. Anderson, *PCI System Architecture*, ch. 5, pp. 59–72. PC System Architecture, MindShare, Inc, fourth ed., 1999.

[9] NCR Corporation, *SCSI: Understanding The Small Computer System Interface*. Prentice Hall PTR, Dec. 1989.

[10] Digital Equipment Corporation, *Writing VMEbus Device Drivers*, June 1997. `http://www.unix.digital.com/faqs/publications/dev_doc/DOCUMENTATION/HTM%L/AQ0R7GTE/TITLE.HTM`.

[11] Intel Corporation, *i960. RM/RN I/O Processor Developer's Manual*, July 1998. `http://www.intel.it/design/iio/manuals/techinfo/80960rmrn/17_arb.htm#25%58`.